GRANT
T.R.
IN-92

2000 002 836

FINAL REPORT

SOLAR FLARE PHYSICS

December 21, 1999

NAG 5-8029 (UMD 01-5-26587)
A Grant from the Goddard Space Flight Center,
National Aeronautics and Space Administration

January 1, 1999 to December 31, 1999

Institution: Astronomy Department, University of Maryland
College Park, MD 20742

Edward J. Schmahl
Principal Investigator
Soc. Sec. No. [redacted]
(301) 405-1506

Mukul R. Kundu
Co-Principal Investigator
Professor of Astronomy

Solar Flare Physics

We have continued our previous efforts in studies of fourier imaging methods applied to hard X-ray flares. We have performed physical and theoretical analysis of rotating collimator grids submitted to GSFC for the HESSI (High Energy Solar Spectroscopic Imager) mission involving the imaging of solar flares in hard X-rays.

In the following pages we present analysis of several different imaging techniques, and focus on the one (The Polar Representation) that shows the greatest promise in minimizing storage requirements and maximizing the speed of map making.

SECTION I

Four Representations of HESSI Visibilities: Which is Optimum?

Four Representations of HESSI Visibilities: Which is Optimum?

We discuss four basic ways the HESSI visibilities can be stored and processed for mapping. In all of these the number and spacing of the time bins selected for the modulation profiles of each HESSI detector have direct consequences for the speed of processing and storage requirements.

The shortest time bins for HESSI will be 0.5 ms. This is small enough to obtain 8 samples per cycle for maps made near the solar limb using the finest collimator. If instead we only ask for 4 cycles per cycle, imaging is not seriously compromised, and the required array sizes are reduced by a factor of 2. Table I shows the number of bins required for each sub-collimator (N_j), given 4 or 8 bins per cycle.

Table 1

No. Bins per Half-Rotation for 8 and 4 per Cycle

Coll	N_j	N_j
	(8/cycle)	(4/cycle)
1	4000	2000
2	2309	1155
3	1333	667
4	770	385
5	444	222
6	257	128
7	148	74
8	86	43
9	49	25
total	9396	4694

In the following, we will adopt the 4 bins/cycle numbers for estimates of array sizes in different representations of the visibilities. All the estimates can be scaled up or down accordingly.

Rectangular Grid Representations

If our goal is to use a uniform grid as support for the visibilities (that is, each time bin corresponds to a pixel of the gridded UV plane), we have to choose a fine enough grid that all points are independent. The innermost UV circle must therefore be large enough that no bins overlap, for if they do, there is no one-to-one correspondence between the UV map and the time bins, and there is a poor representation of the UV circle. (This is the greatest weakness of the rectangular representation.) It is easy to calculate that for 25 time bins (column 3 in Table 1), the finest grid that has no overlapping UV points has a radius of 9 pixels for the innermost UV circle. The UV coverage is shown in Figure 1, and a zoomed version shows the innermost circles. The radius of the largest UV circle is 81 times the radius of the smallest circle, So for the binnings of column 3, the radius of the largest UV circles would have to be 729 pixels, respectively. If the visibilities are to be put into square arrays suitable for IDL's FFT, the arrays would have to be big enough to contain the largest UV circle. Using the next power of 2 exceeding 2×729, the arrays would be 2048×2048.

Figure 1

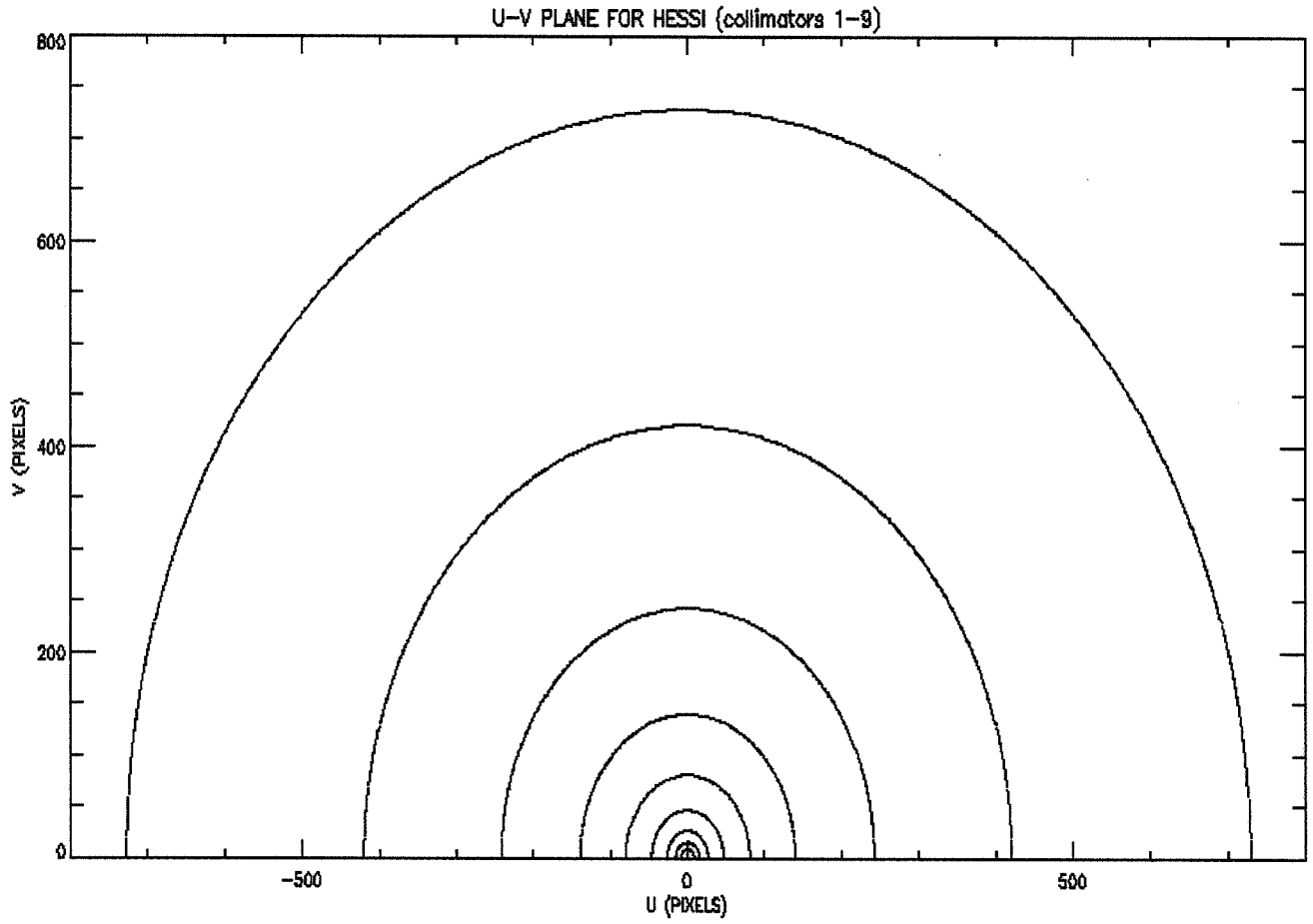
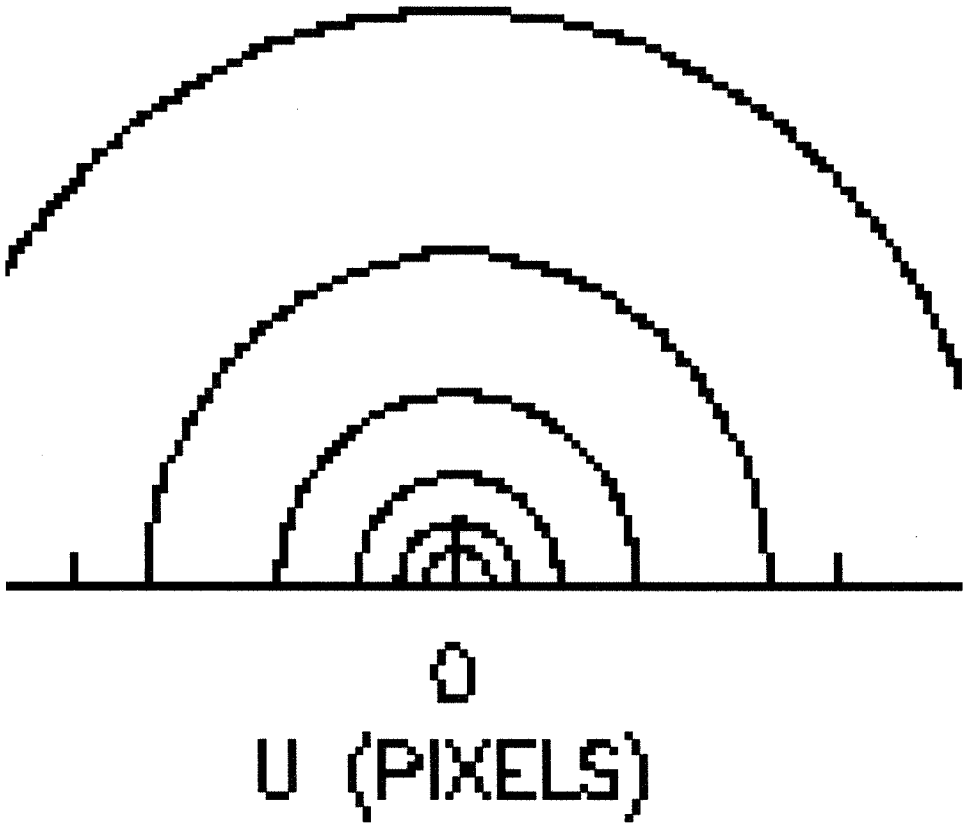
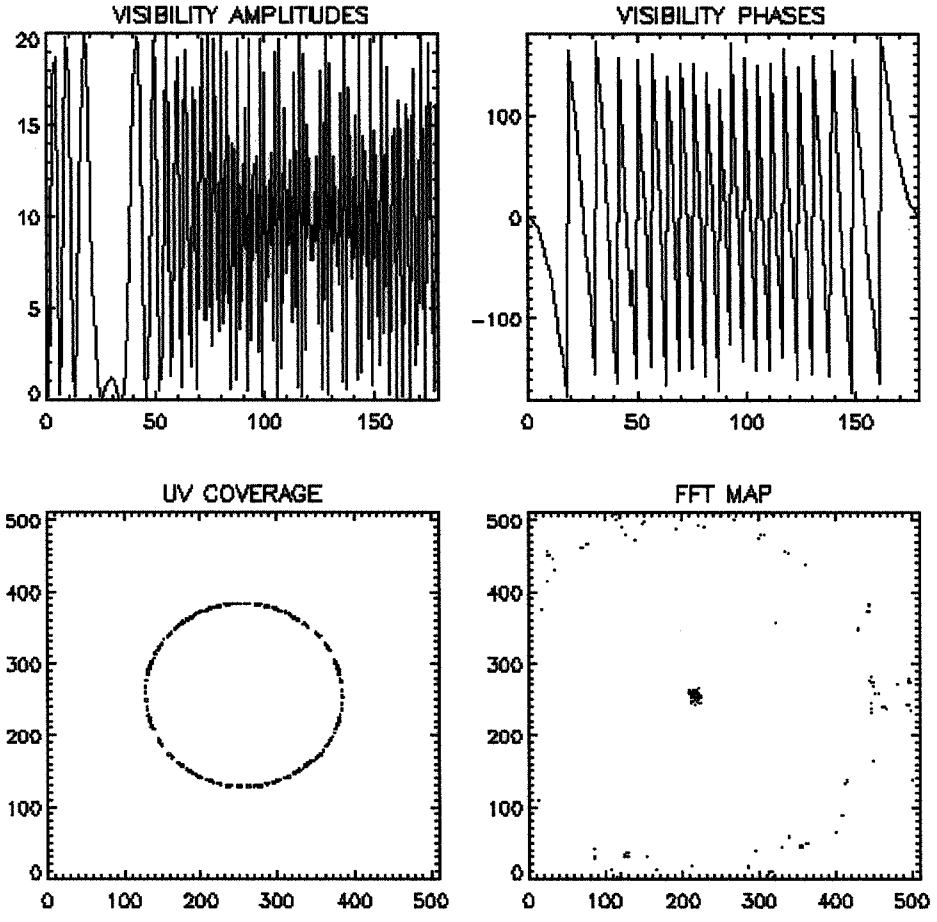


Figure 2



Clearly, these arrays would be very sparse, and careful management of memory would be important if we used this representation. It is probably not efficient to use the standard 2D FFT for such arrays. (In IDL, the FFT of a 2048×2048 complex array is computationally equivalent to about 3×10^8 floating point multiplications.) Another problem with this representation is that the inner circles are not rendered very faithfully by nearest-neighbor sampling of circles, and some kind of interpolation might be necessary.

Example of visibilities and FFT Map for a point source at (600",200")



Direct Representations

Since the arrays described above are sparse, it makes sense to try to deal with only the 1-D arrays corresponding to the 9 UV circles themselves. This strategy, however, has its own inefficiencies. For example, if one wanted to use a Fourier transform of the visibilities to make a map, one would have to compute the quantities:

$$g(x,y) = \sum_k f_k e^{2\pi i(u_k x + v_k y)} \qquad (1)$$

for all the (u_k,v_k) points and all the (x,y) map points. The minimum number of x values required for 9-collimator maps is about 360, corresponding to one pitch of the coarsest collimator and 1'' pixels. Using a brute-force method, a map would require $(360)^2 \times 4694 = 6 \times 10^8$ calculations of the exponentials in the above equation. Rough timing measurements in IDL show that this is about 80 times slower than computing the FFT of a 2048×2048 complex array. So, although we have reduced memory requirements by about a factor of 129 ($4096^2/360^2$), we have lost a factor of 80 in time.

Considerable savings in storage and computations, fortunately, can be gotten by exploiting the facts that the exponentials in equation (1) need only be computed along on one row and one column of the map, and that the map's rows and columns are equally spaced, so one may compute the other entries by recursive multiplication by a constant phasor:

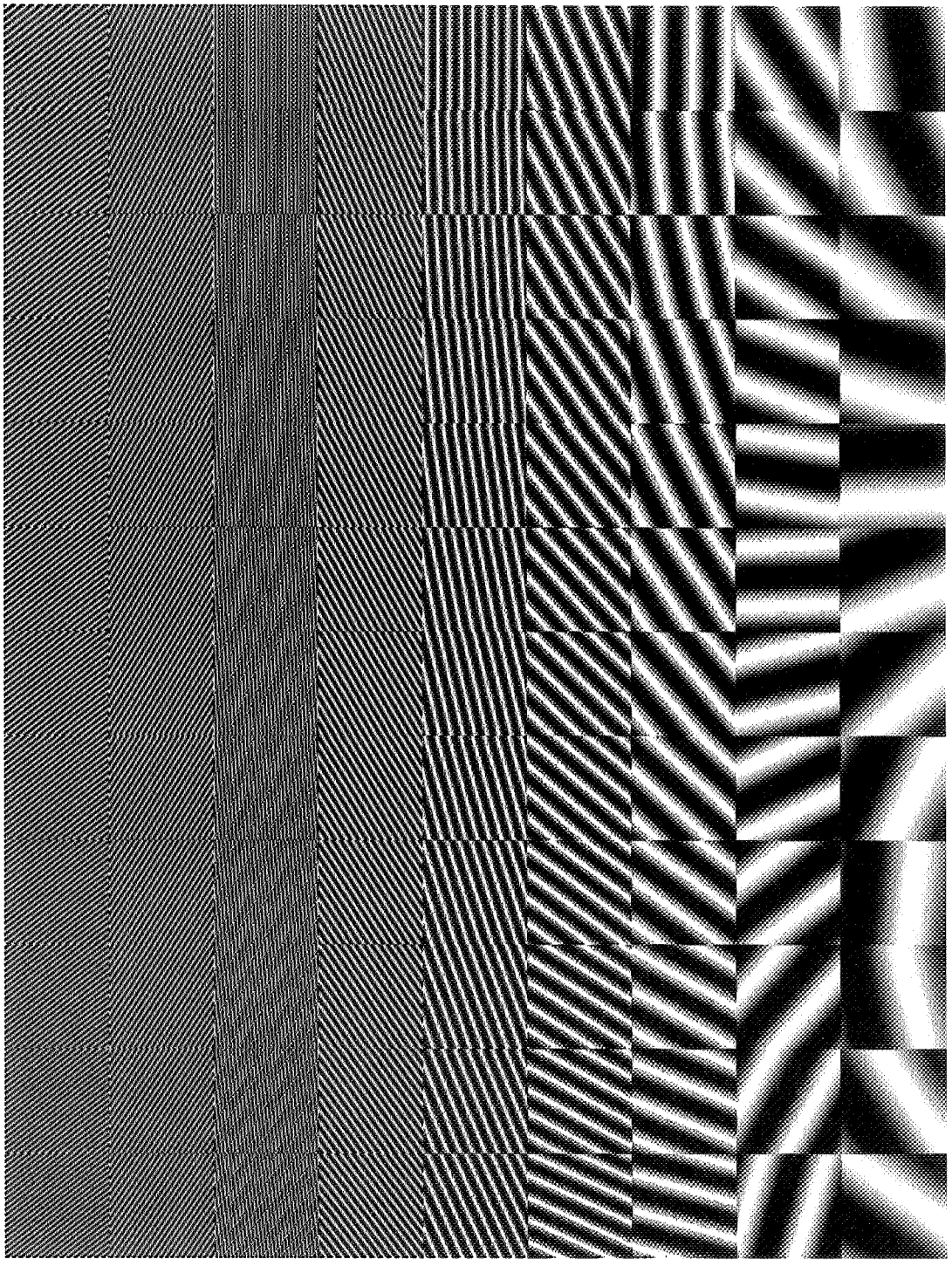
$$e^{2\pi i(u_k x_n + v_k y_m)} = e^{2\pi i u_k x_0} \left(e^{2\pi i u_k \Delta x}\right)^m e^{2\pi i v_k y_0} \left(e^{2\pi i u_k \Delta y}\right)^n$$

This reduces the storage and computational requirements by a factor proportional to the map dimension, M. In practice, IDL tests with M=360 give a speedup by a factor of about 3.5. The overall speed would still be slower than the rectangular (FFT) method.

HXT-Style Modulation Patterns

The HXT style of operation is to use individual modulation patterns for each and every every time bin. This is the approach we have used in the HESSI back-projection and CLEAN programs. The main advantage of the method is simplicity, since there is no need to go into the complex plane or to compute complex exponentials. One disadvantage of this method is that it requires a large amount of intermediate storage, since there must be as many modulation patterns (maps) as there are visibilities (bins). As we show below, however, the HXT-style representation requires an almost unmanageably large amount of storage if all the modulation patterns must be in memory.

An example of some HESSI modulation patterns:



And so on....

Polar Representations

Another strategy is to go into polar coordinates. We show that this gives savings in both time and storage. (It is important to note that the method does *not* assume that the spin axis remains fixed!) We must convert the Fourier transform from rectangular coordinates (*x*,*y*) to polar coordinates (r, θ) in map space, and (*u*,*v*) to (q, ϕ) in visibility space. The rectangular form is:

$$g(x,y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(u,v)\, e^{2\pi i(ux+vy)} du dv \qquad (2)$$

The conversion from rectangular to polar coordinates can include offsets from a nominal center:

$$x = r\ cos(\theta) + a(\phi)$$

$$y = r\ sin(\theta) + b(\phi)$$

where a and b can be arbitrary functions of ϕ (derivable from the HESSI aspect solution). Using this, equation (2) becomes:

$$g(r,\theta) = \int_0^{\infty} q\ dq \int_0^{2\pi} d\phi\ f(q,\phi)\, e^{iq\alpha(\phi)}\ e^{2\pi irq\ cos(\theta-\phi)} \qquad (3)$$

and the phase factor α represents the phase offset from the nominal rotation axis:

$$\alpha = 2\pi(a\ cos(\phi) + b\ sin(\phi)).$$

As a function of radius q, $f(q, \phi)$ is a sum of Dirac delta functions, so the radial integrals are trivial, and we are left with 9 azimuthal integrals:

$$g(r,\theta) = \sum_{j=1}^{9} q_j \int_0^{2\pi} d\phi \; f_j(\phi) \; e^{iq_j\alpha(\phi)} \; e^{2\pi i r q_j \; cos(\theta-\phi)} \qquad (4)$$

The visibilities' azimuthal dependence can be represented as a Fourier series with the number (N_j) of (complex) coefficients given by column 2 or 3 in Table 1. N_j is large when j=1 and is 81 times smaller when j=9:

$$f_j(\phi)e^{iq_j\alpha(\phi)} = \sum_{k=-N_j}^{N_j} f_{jk} \; e^{ik\phi} \qquad (5),$$

where N_j is the number of visibilities for the *j th* subcollimator.

Plugging (5) into the Fourier integral (4) above gives:

$$g(r,\theta) = \sum_{j=1}^{9} q_j \sum_{k=-N_j}^{N_j} f_{jk} \int_0^{2\pi} d\phi \; e^{2\pi i r q_j \; cos(\theta-\phi)+ik\phi} \qquad (6)$$

Then using the integral representation of the Bessel function J_k, we are left with:

$$g(r,\theta) = \frac{1}{2\pi} \sum_{j=1}^{9} q_j \sum_{k=-N_j}^{N_j} f_{jk} \; J_k(2\pi r q_j) \; e^{ik(\theta+\pi/2)} \qquad (7)$$

Here we have 9 Fourier transforms of the $f_{jk}J_k(2\pi r)$ arrays. These arrays are not square, and will be of size $2N_j \times M$,where N_j is the number of points on the *j th* UV circle and M is the map size. These arrays are somewhat smaller than in the direct representation (Sect. 3), but they scale the same way. For the case M=360, the arrays will be about 4 times smaller, and are smaller by even larger factors than the arrays in the rectangular representation.

Furthermore, all the Fourier transforms can be done using FFTs, because ϕ is distributed uniformly around the UV circles. (In the direct representation the u_k and v_k are not evenly spaced in azimuth, making FFTs impossible.) So the polar approach has potential advantages for both speed and storage.

It is worth looking for a way to speed up the Bessel function calculations, since they are the limiting factor here. In fact, a factor of 2 speedup can be gotten by computing only the even order Bessel functions, and using the recurrence formula for the odd orders:

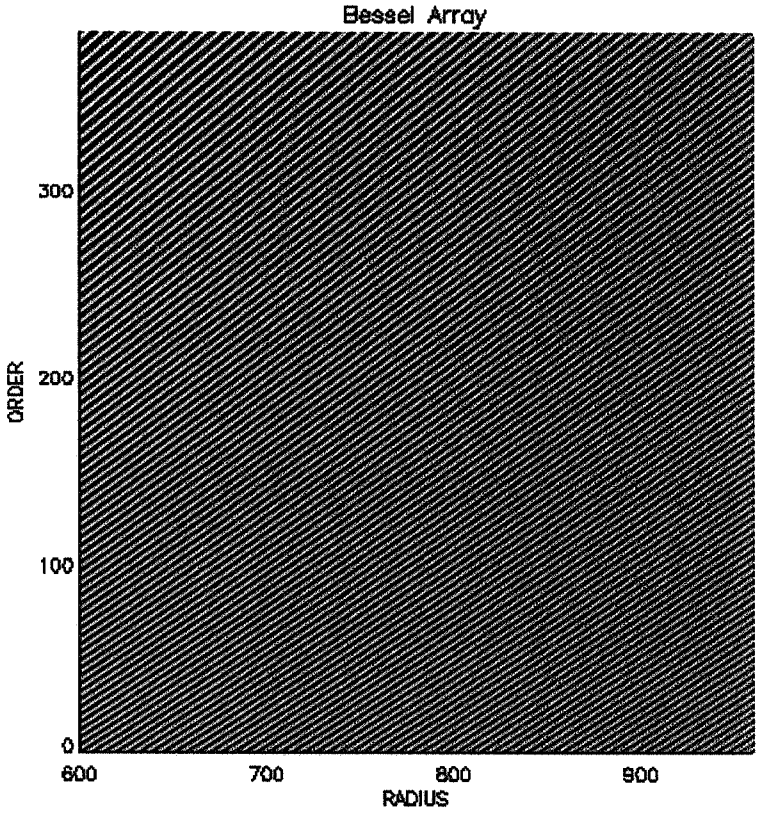
$$J_{\nu-1}(z) = (z/2\nu)(J_{\nu-1} + J_{\nu+1})$$

Extensions of this trick lead to about 2 orders of magnitude speedup of the Bessel function calculations.

IDL tests show that it is possible to use this ''downward'' recurrence relation to produce an array of Bessel functions about 50 times faster than the stock IDL routine. Effectively each of the Bessel coefficients in equation (7) can be produced in a few floating-point operations, instead of the estimated 450 for repeated calls to IDL's beselj(x,n). In practice, one calls beselj(x,n) only twice, and creates each of the remaining Bessel functions by one multiplation, one division, and one addition. Tests show that this is a very stable recurrence.

One minor disadvantage of using polar coordinates is that in projected solar coordinates the maps will be bounded by non parallel lines and circular arcs, but this can be mitigated by interpolating them onto square arrays after the image processing is complete.

An example of an array of Bessel Functions



Summary

The storage requirements for the four representations are wildly different in many respects. For purposes of comparison, we use M=360 for the map size and N=4694 for the number of visibilities. The first half of Table 2 shows the storage space (in floating point numbers) as a function of N and M. The second half shows the corresponding actual numbers.

Table 2

Storage Requirements for Each Approach (All 9 Collimators)

	Rectangular	Direct	HXT Style	Polar
Visibilities:	$2N+1$	$2N+1$	N	$2N+1$
Intermediate:	$\sim 0.2N^2$	$4MN$	M^2N	$2MN$
Map pixels:	$\sim 0.2N^2$	M^2	M^2	M^2
Visibilities:	37 KB	37 KB	18.7 KB	37 KB
Intermediate:	34 MB	54 MB	2433 MB	13 MB
Map pixels:	34 MB	0.5 MB	0.5 MB	0.5 MB

The number of equivalent floating-point operations for one transform from visibility space to map space is readily estimated. The table shows the ''brute force'' numbers for the direct and polar methods as well as the ''recurrence'' numbers described in Sections 3 and 5. The former uses the relative timing (in IDL 5.2) for Bessel functions, FFT and exponentials: beselj(d,100) : FFT(d,1) : exp(2*pi*i*d*x) : d*x = 450 : 54 : 49 : 1. When arrays of exponentials or arrays of Bessel functions are calculated using recurrence, both the exponential and Bessel function computation times can be reduced to about 12 multiplications.

Table 3

Computational Requirements for Each Approach (All 9 Collimators)

	Rectangular	Direct	HXT Style	Polar
		BRUTE FORCE:		
Equiv. FP ops:	$14*N^2$	$50N\,M^2$	$50N\,M^2$	$500\,N\,M$
Equiv. FP ops:	3.e+08	3.e+10	3.e+10	8.e+08
		RECURRENCE:		
Equiv. FP ops:	-	$?*N\,M$	-	$20\,N\,M$
Equiv. FP ops:	-	9.e+09	-	4.e+07

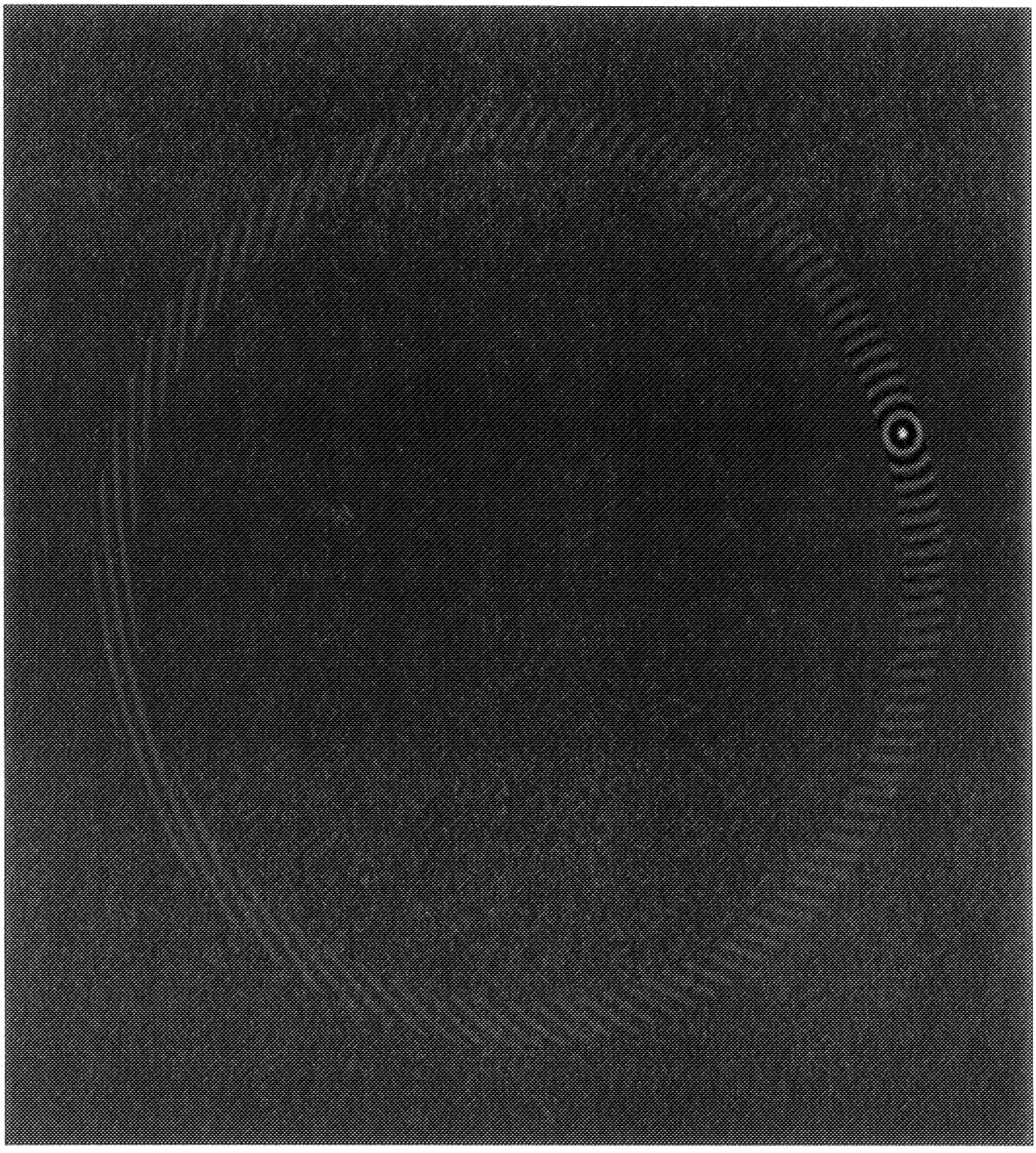
These numbers should be considered accurate to about a factor of 2. Clearly the HXT-style representation is a major loser in computational speed, and with the use of recurrence, the polar representation is the winner.

Since the HXT-style representation scales as M^2, smaller maps could be made faster. A $\mathbf{60 \times 60}$ map, for example, would be made in only slightly more time than in the rectangular representation. But the storage requirements (67 MB) would still be larger than in the other methods.

Recipe for a Polar Map

1. Obtain the amplitudes and phases from the count rates for half a rotation, creating 9 vectors of phasors $f_j(\phi)$. These vectors will range in length from 25 up to 4000 elements.
2. Compute the Inverse FFT in ϕ space for each complex vector $f_j(\phi)$,and get the complex coefficients f_{jk} (See eq. 5).
3. Using downward recurrence relations, compute the 9 matrices $J_k(2\pi r\ q_j)$. For the coarsest collimator, this is a $\mathbf{25 \times 360}$ array, and for the finest, it is $\mathbf{2000 \times 360}$.
4. For each radius (r) in the map and each collimator (j), compute the vector $f_{jk}J_k(2\pi r\ q_j)\ e^{ik\pi/2},\quad k = -N_j, ..., N_j$.
5. Obtain the one-dimensional FFT, giving the θ dependence at that radius (one column of a one-collimator map).

6. Add the one-collimator maps together, weighting by the UV radius q_j. This is the natural weighting, but alternatives are possible.

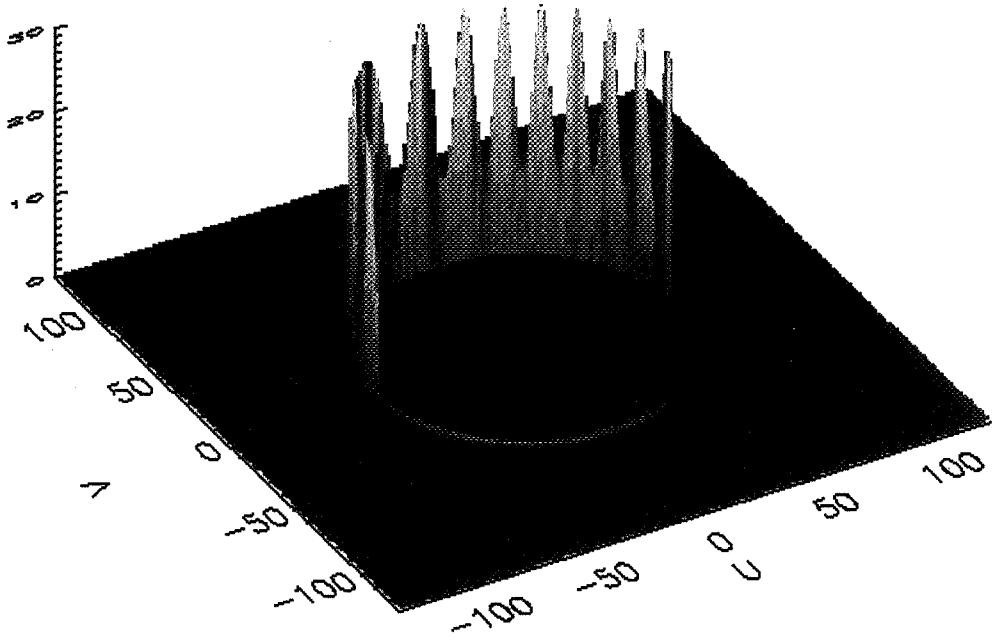


SECTION II
Countrates to Visibilities to Polar Maps

Countrates to Visibilities to Polar Maps

The countrates (modulation profiles) from each HESSI detector, when properly calibrated, will provide Fourier information that can be used to reconstruct an image. Each of the nine modulation profiles represent components of the Fourier transform (visiblity) of the image, with support consisting of circles in the Fourier (UV) plane. Figure 1 shows a representative profile plotted (for one half rotation of HESSI) above one semicircle in the UV plane.

Figure 1a: Modulation Profile on UV Circle



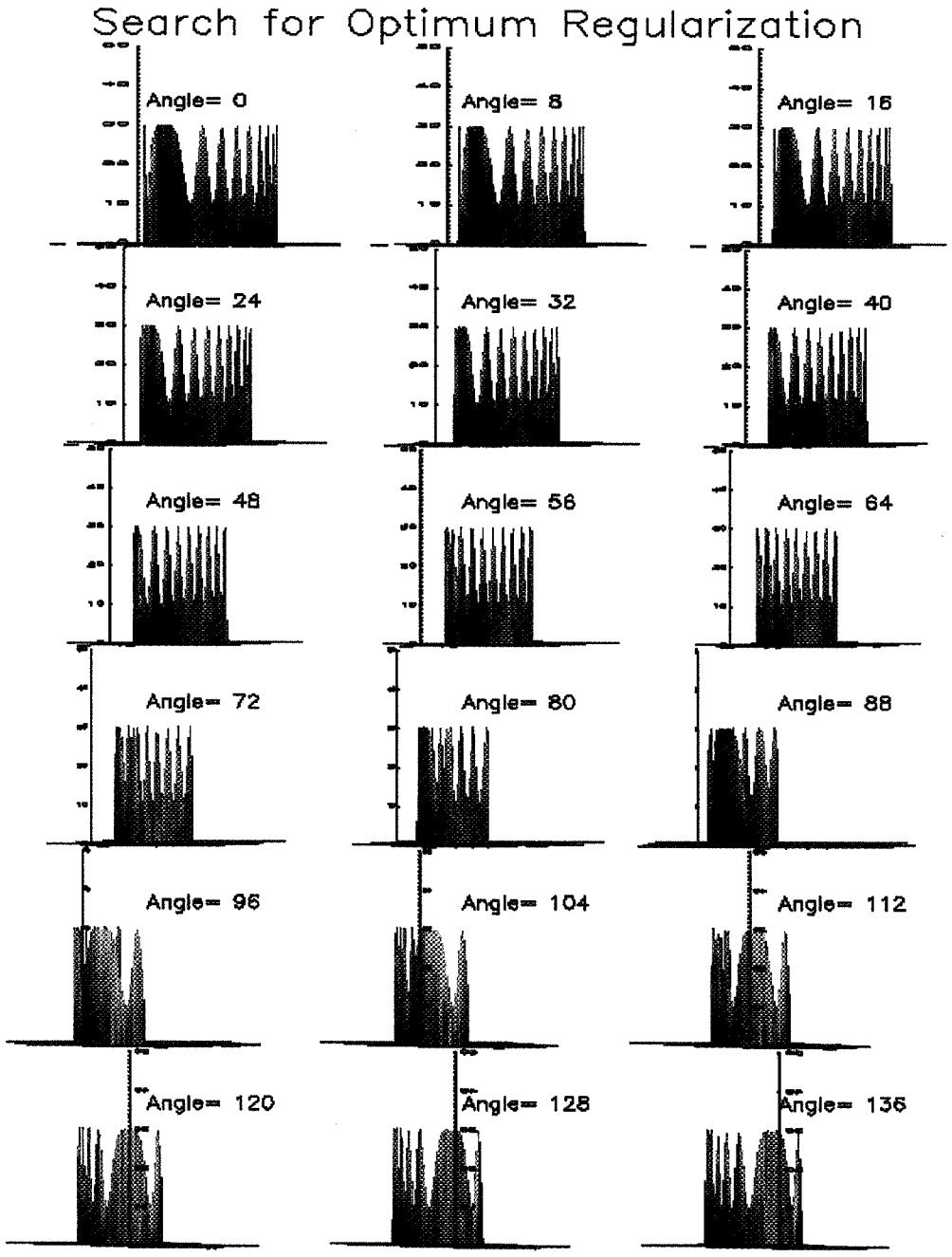
Before one can convert the (real) modulation profile into (complex) visibilities, it is necessary to find the phase center about which the map will be constructed. This can be done in several fundamentally equivalent ways:

- The phase center corresponds to a weighted centroid of the flare in inertial coordinates, and it corresponds to the best point-source fit to the flare. One could find that point by making back-projection maps with the coarsest collimators, and simply finding the maximum.
- It is also possible to find the point by counting the number of modulation cycles--which gives the distance from the spin axis in units of the angular pitch--and finding the time of slowest modulation--which gives the azimuth of the source.
- From the standpoint of signal processing, the azimuth of the phase center may be found by "regularizing" the argument (time or angle) of the modulation profile. The regularized coordinate is a sinusoidal function of time, and the optimum regularization has a phase which gives the azimuth of the phase center.
- There is also a geometric interpretation of the phase center which we illustrate here. The phase center can be shown to lie on that line through the spin axis from which the projected modulation

profile of Figure 1 appears to be the most regular. Figure 2 shows a range of views of Figure 1. At only one of these views do the profile's maxima appear at regular spacing.

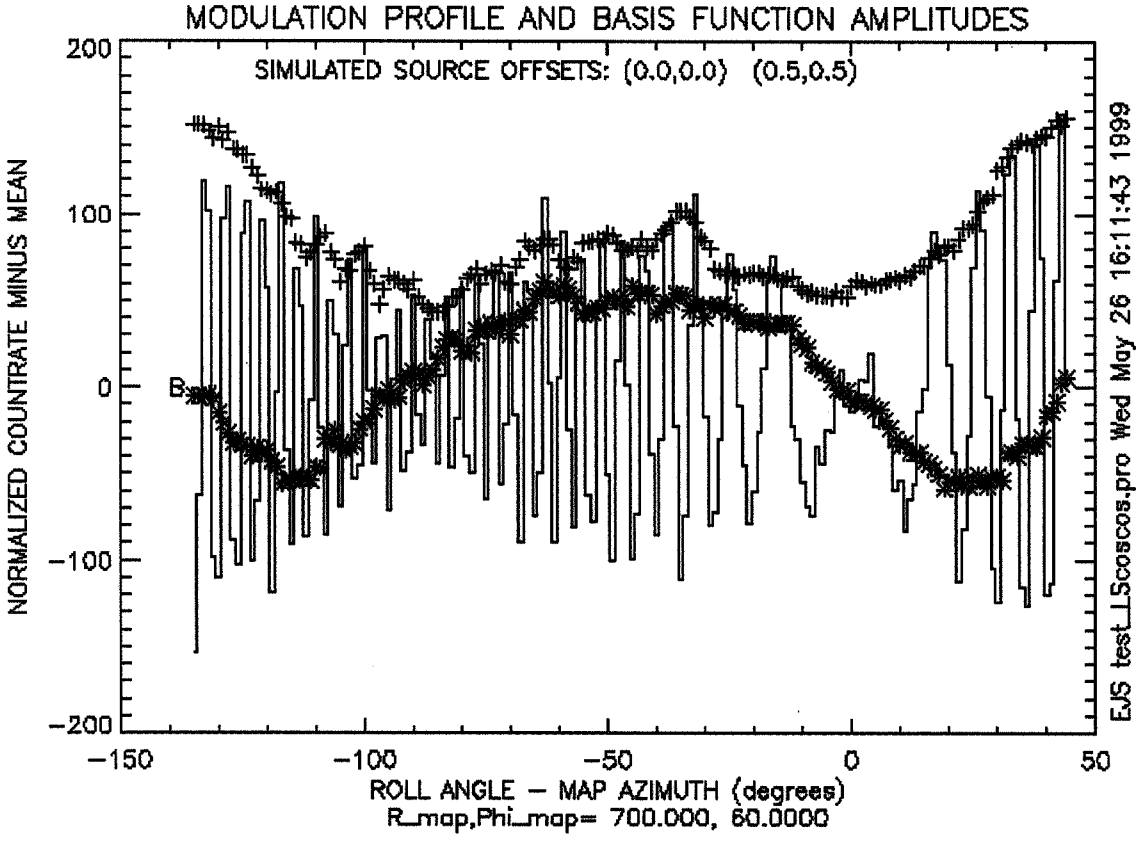
It should be noted that in all of the above methods there is a 180-degree azimuth ambiguity. This can be resolved by multi-rotation averages in the presence of axis drifts, or from context data.

Figure 1b:



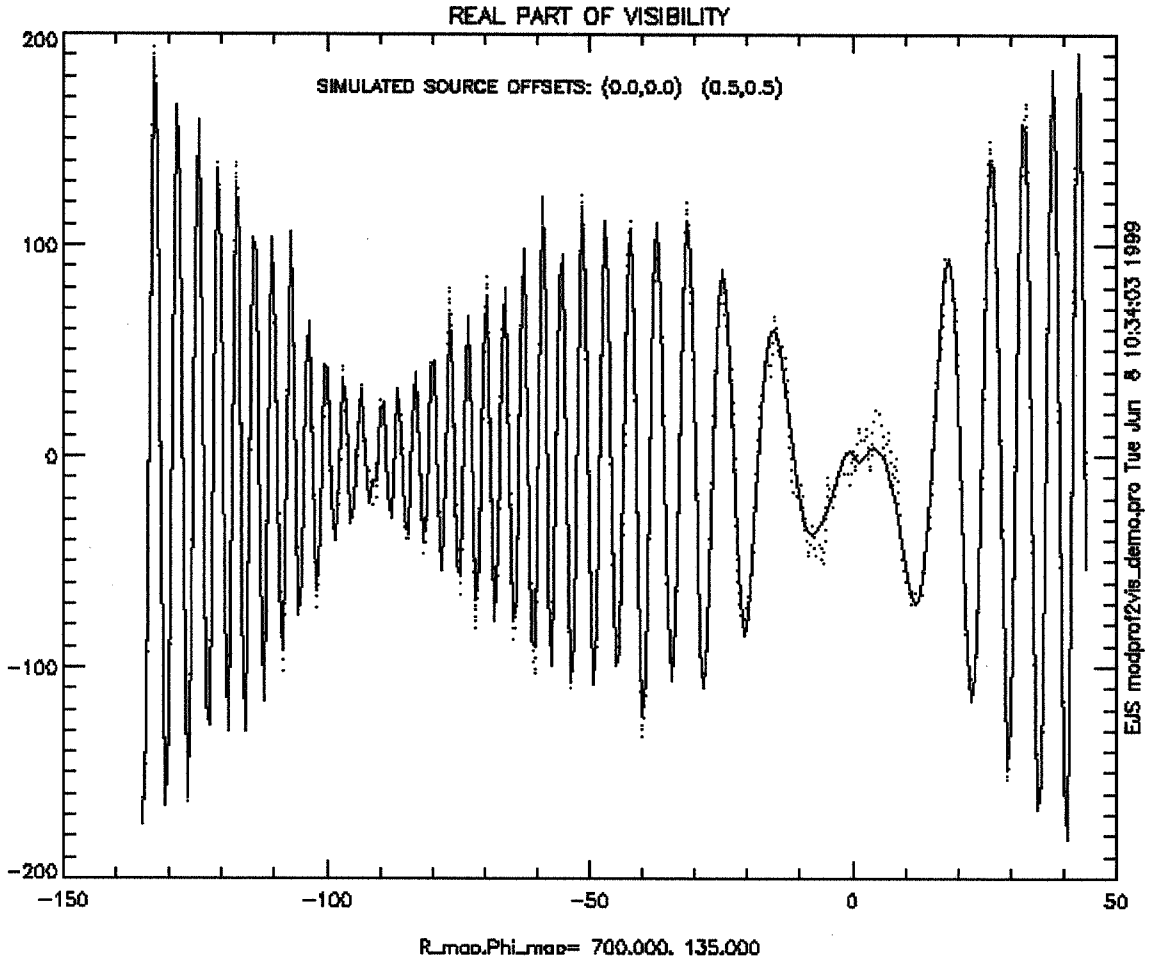
Once we have found the phase center, one may regularize the modulation profile and fit sinusoids to each of the modulation cycles. The amplitudes and phases of these sinusoids, when pieced together, provide the visibilities of the source. Alternatively, one may fit the profile with functions of the form cos(A*cos(B*t)+C), which are the "basis functions" for point source time series. (This has the advantage that the time intervals around the times of slowest modulation, where sinusoids of time t do not need to be treated separately.) This entire procedure makes the crucial assumption that the amplitudes do not vary significantly during one modulation cycle. (Of course, if the source changes on time scales much less than 2 s, the entire imaging process has to be revised.)

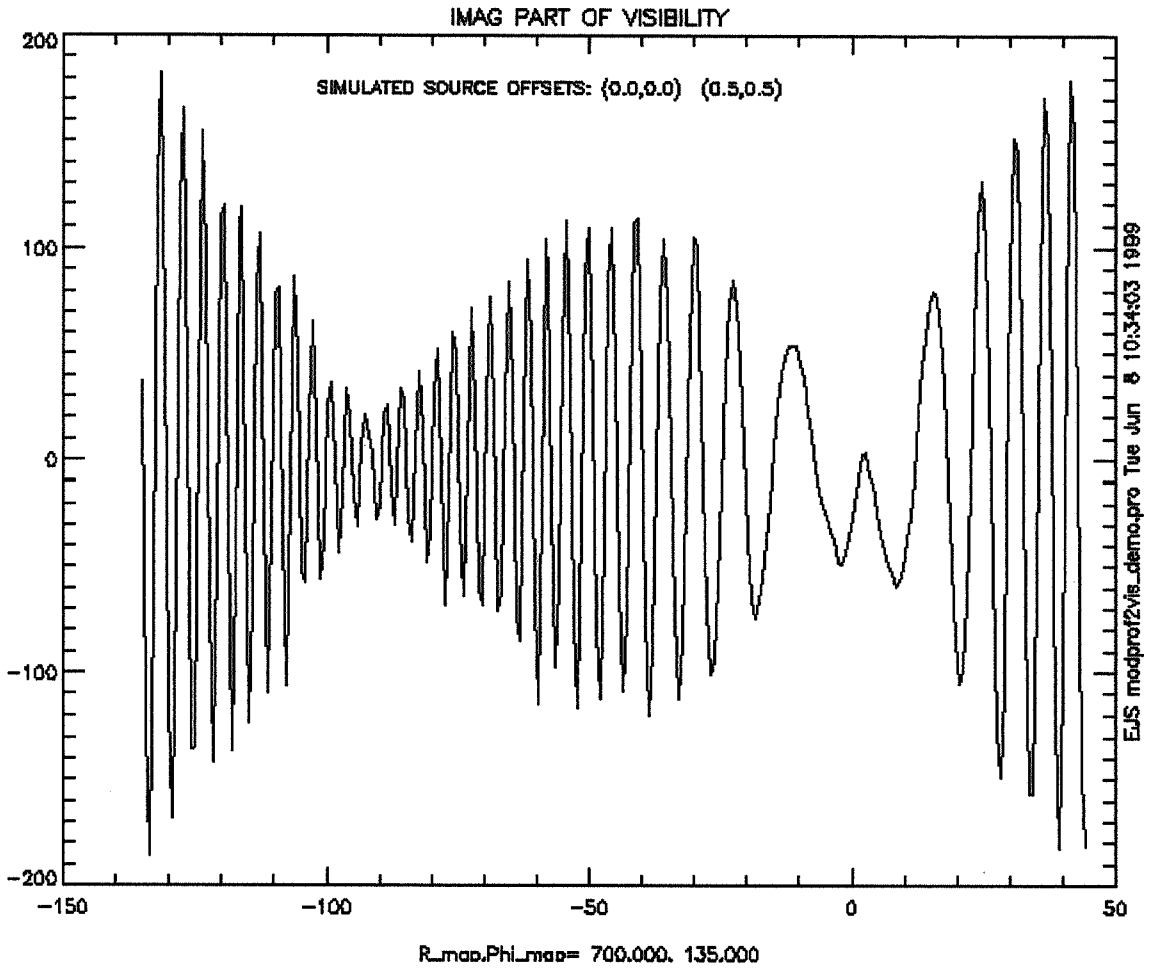
Figure 1c: Modulation Profile plus Basis-Function Amplitudes



Piecing the basis-functions together provides the visibilities, as shown in the following figures. The dots on the real plot are the original modulation profile values.

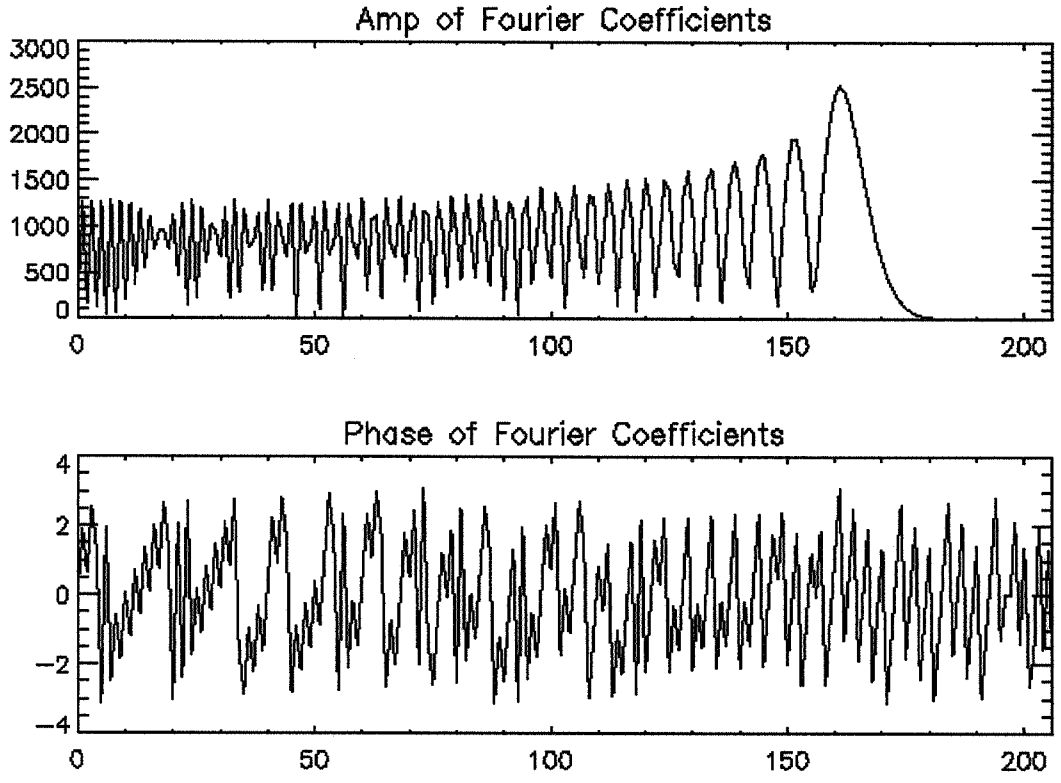
Figure 1d: Visibility





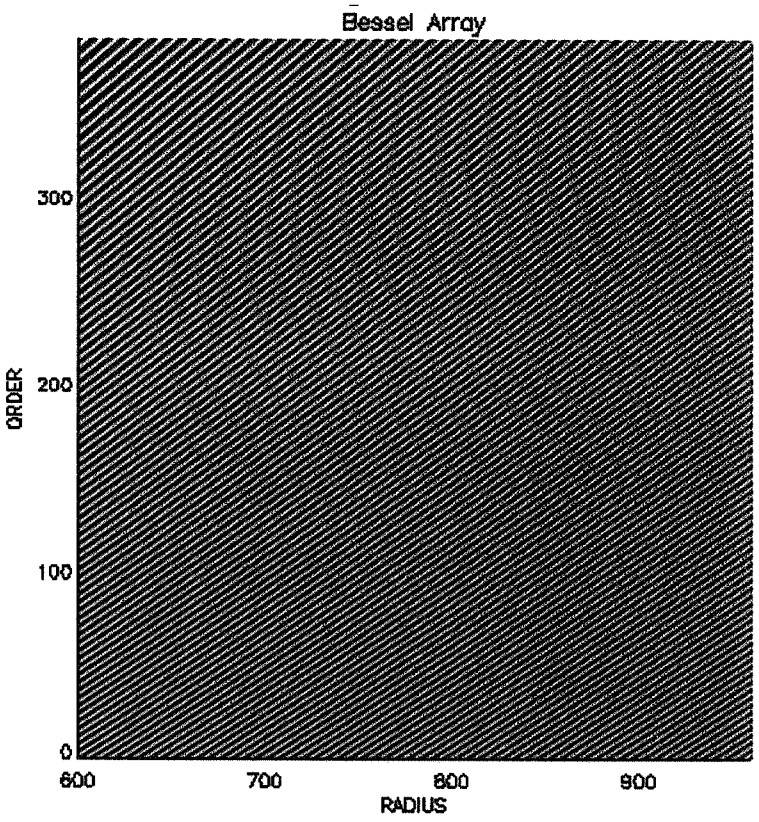
As indicated in the **recipe for polar mapping** (.gif, .ps), the polar map of the visibilities requires an initial Fourier transform (one-dimensional!) of the visibilities, and these are shown in the following figure.

Figure 2: Amp and Phase of Fourier Coefficients



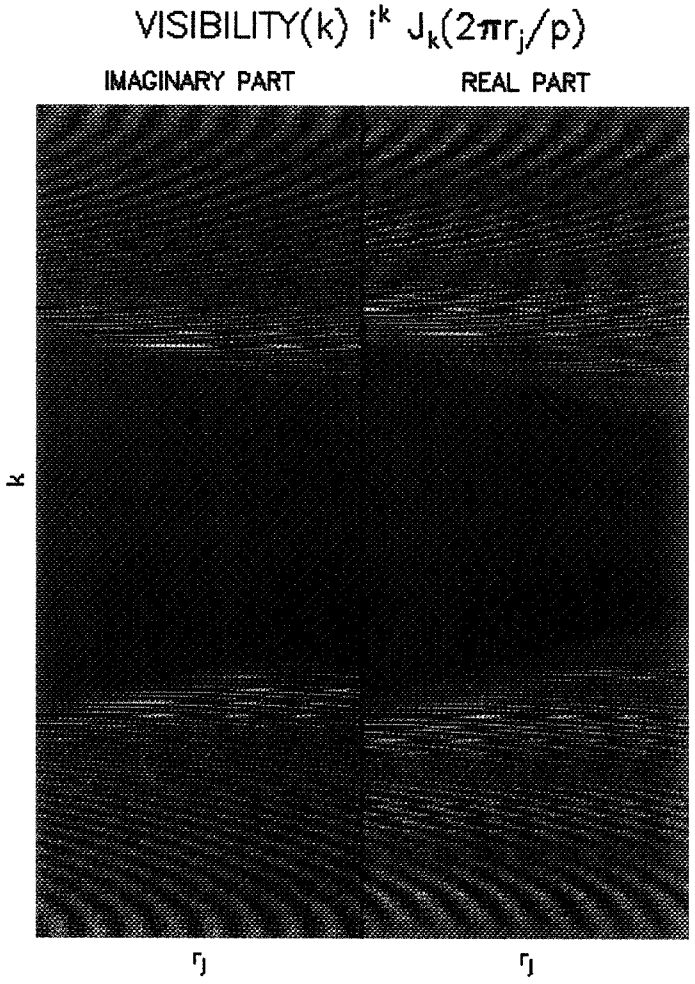
Then one multiplies the above transform by each column of a array of Bessel functions:

Figure 3



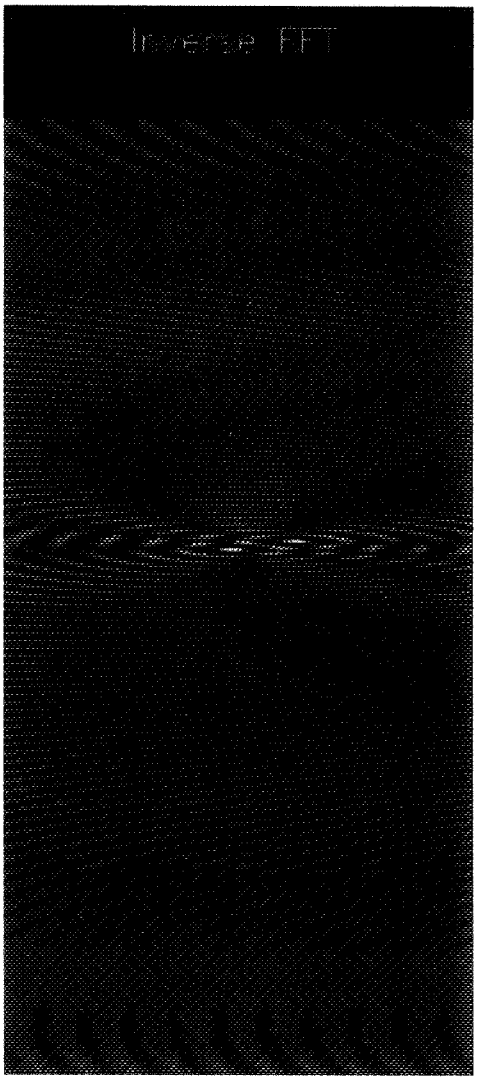
And the resulting product is a transform of the polar map:

Figure 4: Visibility $\times i^k \times J_k$



Each column of the above array must be back-transformed to obtain the polar map. This is a about factor of 2 faster than a full 2-D transform, but what makes the method fast is the fact that the width of the above arrays is only the final map width, and not the 2048 required by sampling for rectangular representations. Finally, we get the polar map, where the figure has radius on the horizontal axis and azimuth along the vertical axis:

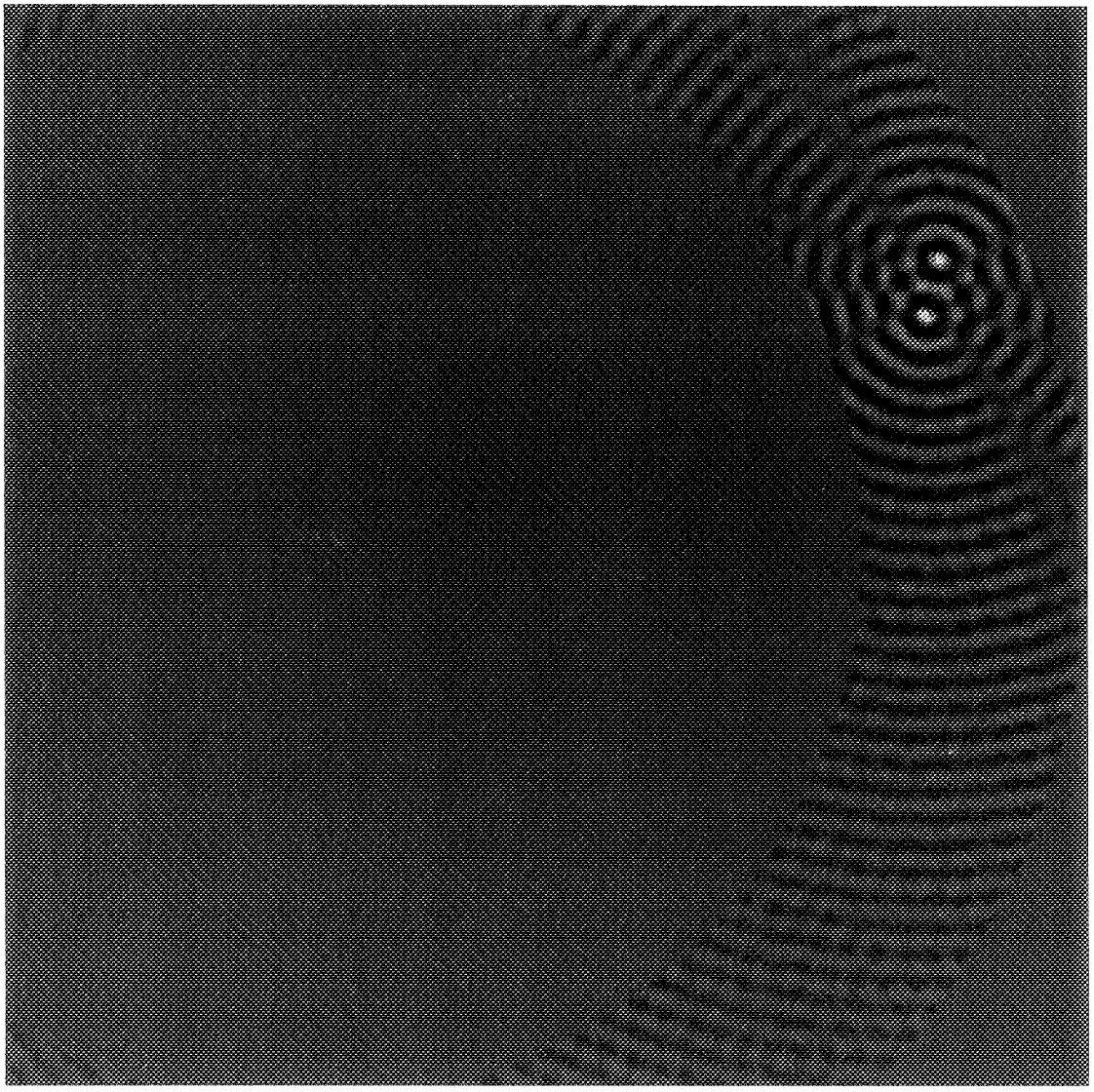
Figure 5: Polar Map



The polar map can then be projected onto rectangular coordinates:

Figure 6

POLAR MAP PROJECTED ON X-Y PLANE



Recipe for a Polar Map

1) Sum all the wavefronts and do a fast Fourier transform, saving a vector of phases $\phi_{\theta}(\omega)$ for each θ.
2) Compute the inverse FFT in ω space for each complex vector $f_{\theta}(\omega)$ and get the complex coefficients $f_{\theta}(\omega_0, \theta)$.
3) Using a monotonic relationship between the θ position $x_{\theta}(\theta)$ and θ. For the nearest millimeter, this is a 90 x 360 array, and for the best it is 1000 x 4000.
4) For each radius ω_0 in the map and each millimeter θ, compute the vector $f_{\theta}(\omega_0, \theta)$.
5) Sum up the one-dimensional FFT, along the θ dependence at that radius (one column of a two-dimensional map).
6) Add the normalized maps together, weighting by the DFT radius ω_0. This is the natural weighting. Other weightings are possible.

---

2. Compute the Inverse FFT in ϕ space for each complex vector $f_j(\phi)$,and get the complex coefficients f_{jk} (See eq. 5).

3. Using downward recurrence relations, compute the 9 matrices $J_k(2\pi r\ q_j)$. For the coarsest collimator, this is a 25×360 array, and for the finest, it is 2000×360.

4. For each radius (r) in the map and each collimator (j), compute the vector $f_{jk} J_k(2\pi r\ q_j)\ e^{ikx/2}$, $k = -N_j, \ldots, N_j$.

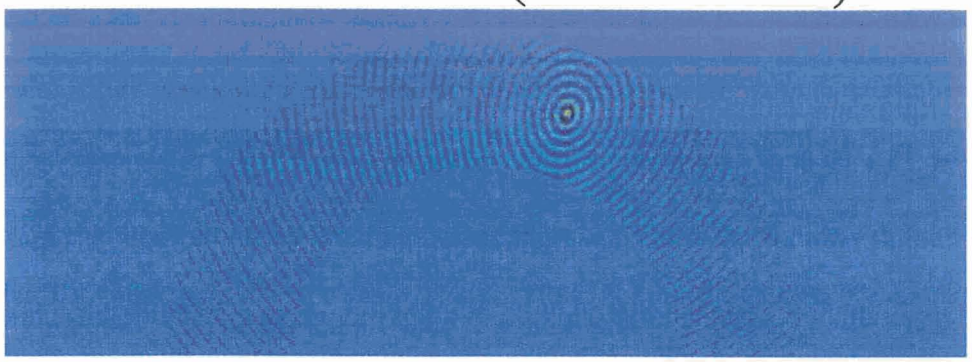
5. Obtain the one-dimensional FFT, giving the θ dependence at that radius (one column of a one-collimator map).

6. Add the one-collimator maps together, weighting by the UV radius q_j. This is the natural weighting, but alternatives are possible.

Ed Schmahl

Ed Schmahl
Last modified: Thu Jun 10 12:46:46 EDT 1999

POLAR MAPPING (October 22, 1999)



Polar Mapping

HESSI is a Fourier-transform imager and its primary data form consists of samples in the Fourier (u,v) plane of the Fourier transform of a flare. There is a rich heritage of Fourier transform imaging, most of it in the radio domain, but most Fourier imaging in astronomy has been done in rectangular, Cartesian coordinates, where the ''dirty map'' $D(x,y)$ is given by the transform of visibilities:

$$D(x,y) = \int_{-\infty}^{\infty} du \int_{-\infty}^{\infty} dv\ V(u,v)\ e^{2\pi i(ux+vy)} \qquad (1)$$

But HESSI is a spinning telescope, and the samples in the U,V plane are on circles $u^2 + v^2 = k^2 = \text{constant}$, one for each subcollimator and harmonic. Therefore it seems to make sense to go into polar coordinates, and it turns out that this is an efficient mapping strategy. (The method does not assume that the spin axis remains fixed.)

We must convert the Fourier transform from rectangular coordinates $(\mathbf{x},\mathbf{y})$ to polar coordinates (r,θ) in map space, and $(\mathbf{u},\mathbf{v})$ to (k,ϕ) in visibility space:

$$D(r,\theta) = \int_{0}^{\infty} k\ dk \int_{0}^{2\pi} d\phi\ V(k,\phi)\ e^{2\pi irk\ cos(\theta-\phi)} \qquad (2)$$

But the integral over the radial wavenumber k becomes a sum over the number of subcollimators (and possibly their harmonics), because the visibility is sampled on

circles:

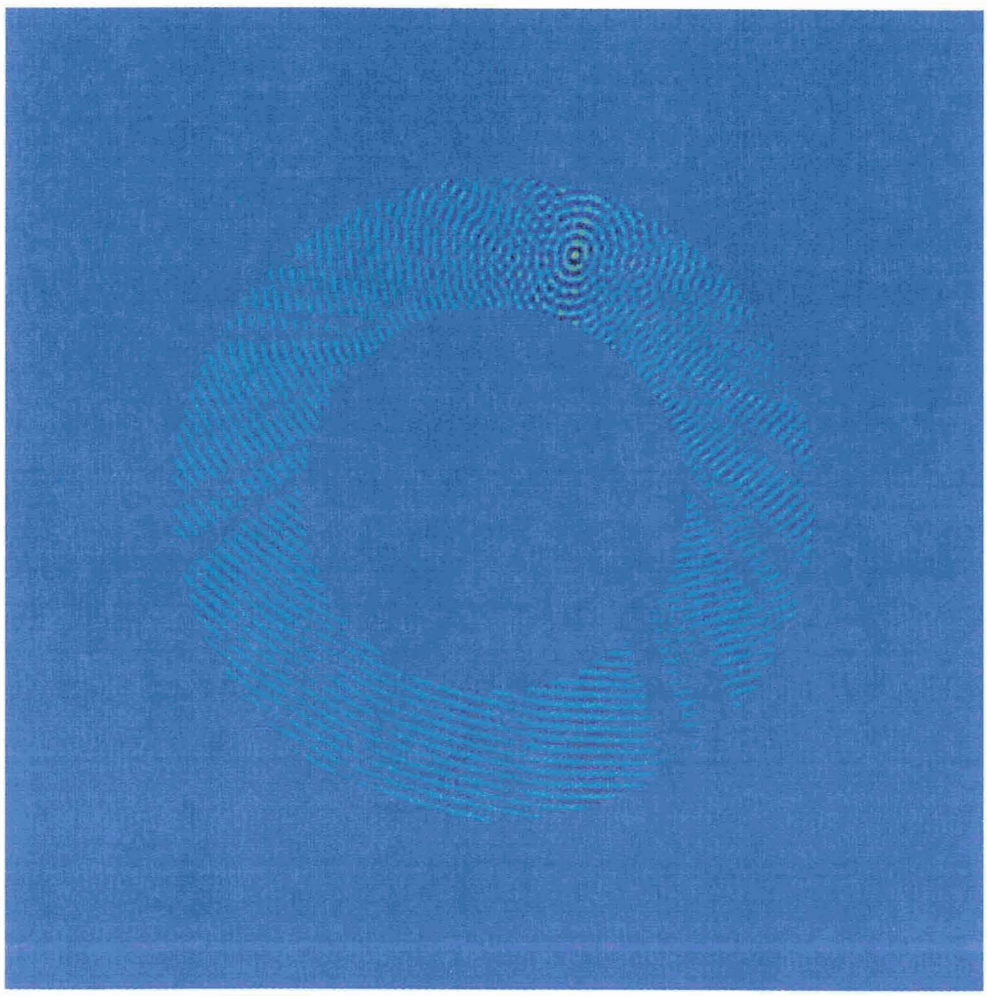
$$D(r,\theta) = \sum_k k \int_0^{2\pi} d\phi \; V_k(\phi) \; e^{2\pi i r k \cos(\theta-\phi)} \qquad (3)$$

The above integral is a convolution, and therefore intrinsically faster to compute and more compact for computation than equation (1). This in a nutshell explains why Polar Mapping for HESSI is more efficient than rectangular mapping: the integrals are one-dimensional, and if the samples of visibility are equally spaced on the (u,v) circles (impossible in rectangular coordinates), FFT methods may be used in an efficient manner.

SIMULATION EXAMPLES

EXAMPLE 1

Polar Map for Detector 3, 1400 time tags

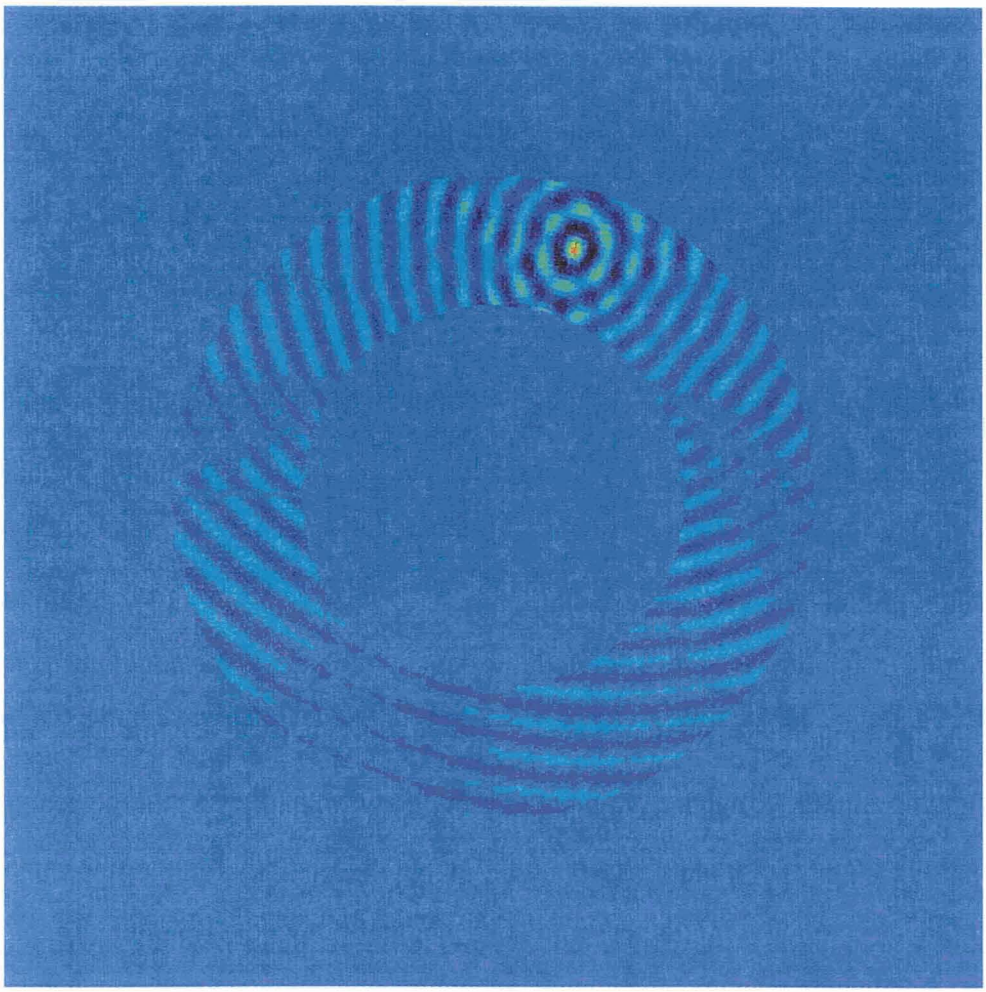


EXAMPLE 2

Polar Map for Detector 4, 1400 time tags

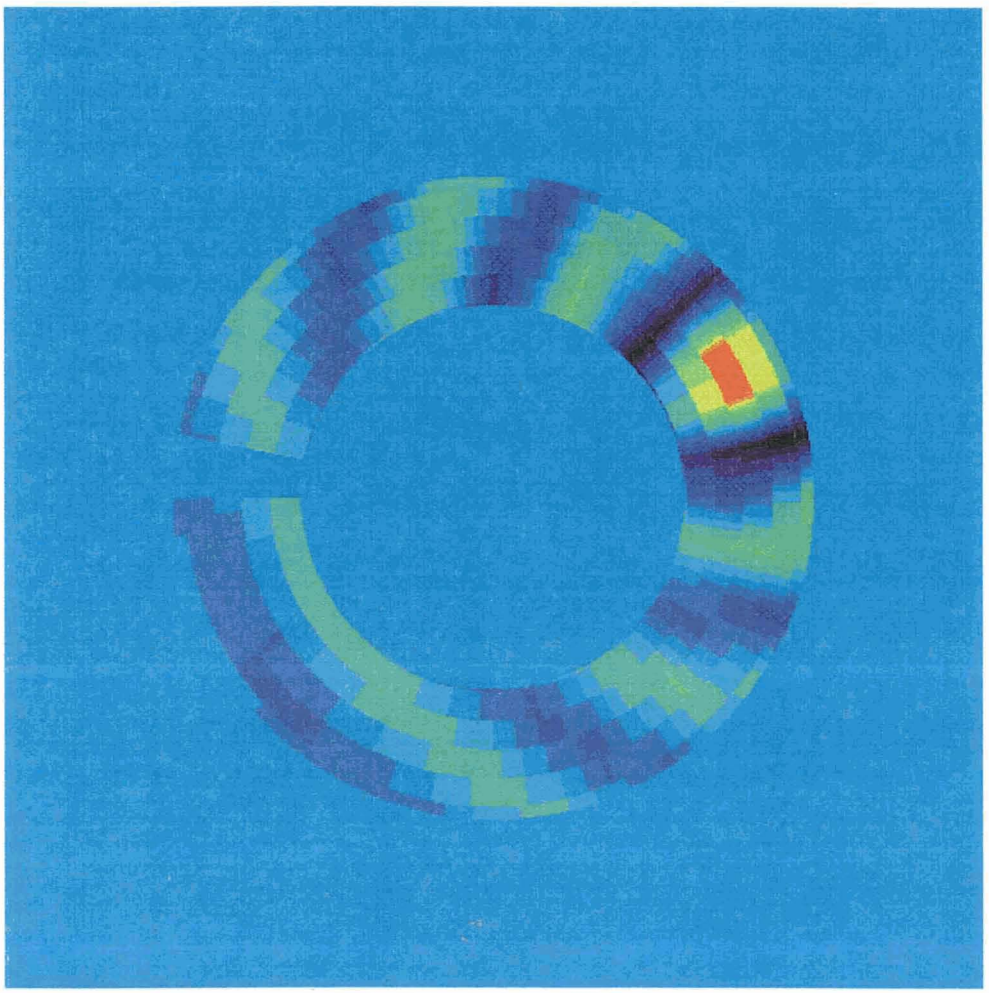
EXAMPLE 3

Polar Map for Detector 5, 1400 time tags



EXAMPLE 4

Polar Map for Detector 8, 98 time tags



PROGRAMS (October 25, 1999)

(WARNING: Bugs exist in these programs, and they are likely to be out of date by the time you read this.)

polar_mapper.pro

```
;+
; NAME:  polar_mapper
;
; PURPOSE:
;        MAKES A POLAR MAP FROM A CALIBRATED EVENT LIST
;
; USAGE:
;         1. Make a model map and call sim_calib_eventlist
;         2. Convert the counts to visibilities using calibevents2vis
;         4. Get the aspect structure for the same time bins
;         5. Shift the visibility phases to sun center
;         6. Make the polar map with vis2pmap
;
; INPUTS:
;         det_index = scalar, 0-27
;         ce        = calibrated event list structure
;         xy_offset = [x,y] vector of inertial coords of map center (arcsec)
;         /plot         Set this if you want plots and images
;
; OUTPUTS:
;         dmap = radius and azimuth map
;         dmap_xy = x,y projection (annulus) of dmap
;
; EXAMPLE:
;;   1. MAKE THE MODEL MAP
;       model=fltarr(64,64)
;       model(31,11)=1
;
;;   2. CREATE THE CALIBRATED EVENT LIST for the selected det_index
;       det_index=5
;       xy_offset=[650,200]
;       sim_xy_offset=xy_offset+25
;       sim_photons=2000
;       det_mask=intarr(27) & det_mask(det_index)=1
;       bin_size=2L^12 ; microseconds  (this is ok except for det_index=0)
;       num_bins=(2L^22)/bin_size
;       time_range=[0,4]  ; If this is changed, polar_mapper won't work
;       o = obj_new( 'hsi_calib_eventlist')
;       timebindef = o->get( /time_bin_def )  ; get the default array
;       timebindef(det_index)=bin_size
;       ce = o->getcalibeventlist(det_index, SIM_TIME_range=time_range, $
;            TIME_RANGE=time_range, SIM_PHOT=sim_photons, $
;            A2D_INDEX_MASK=det_mask,TIME_BIN_DEF=timebindef,SIM_MODEL=model, $
;            SIM_XYOFFSET=sim_xy_offset, XYOFFSET=xy_offset)   ;,/saszero
;;   3. Call the mapper:
;       dmap=polar_mapper(det_index,ce,xy_offset,/plot)
;
```

```
;
; RESTRICTIONS:
;       Works only for time_range=[0,4] and for a single subcollimator.
;       If the map position is several pitches away from the flare centroid,
;       the map will be poor.
;
; CALLS:
;                  hsi_grid_parameters
;                  calibevents2vis, (converts counts to visibilities)
;                     ls_sin_cos
;                     ls_amp_pha_ctr
;                  vis2pmap, which makes a polar map calling:
;                     convl (the key to speed)
;                     pmap2xy (interpolates the polar map onto xy coordinates)
;
; VERSION HISTORY:
;        VERS. 1
;          Revised to match compute_model_ce.pro Sept 17, 1999
;        VERS. 2              Sept. 27, 1999 -- EJS
;          Various bugs squashed
;          Forced visibility to be Hermitian
;        VERS. 3
;          Replaced Bessel-fcn array with FFTs
;          Added CONVL program for fast convolution (extends convolve.pro).
;          Made to work with an externally-produced calib_eventlist
;          Lots of bugs remain.                    Oct 26, 1999 -- EJS
;
; NOTES:
;        The grid_angle is used when correcting for aspect, but not in
;           mapping, so the azimuth of the source comes out wrong.
;        No general times yet: We extract only the first 1/2 of 1
;           rotation from an entire rotation.
;
;        Maps are in the wrong hemisphere (S instead of N), probably
;           the result of negatively increasing roll angles.
;        There is something suspicious about the first dark ring in array
;           dmap_xy -- the scaling may be wrong
;
;-

function polar_mapper,det_index,ce,xy_offset,plot=plot
;
;
; RESTRICTIONS:
;       WORKS ONLY FOR TIMERANGE=[0,4]
;

      pitch=((hsi_grid_parameters()).pitch)(det_index)
      grid_angle=((hsi_grid_parameters()).orient)(det_index)

if keyword_set(plot) then plot,ce.dx,ce.dy,tit='ASPECT VALUES'

;  CONVERT THE COUNTS TO VISIBILITIES:

   message,'Running calibevents2vis',/cont
   vis0=calibevents2vis(det_index,ce, cycle_factor=2)  ; best cycle_factor


;  GET THE PHASE FACTOR FOR MAP PHASE CENTER

  alpha=2*!pi*((ce.dx)*cos(grid_angle)+(ce.dy)*sin(grid_angle))/pitch
```

```
        ; there may need to be a sign change


;  SHIFT THE VISIBILITY PHASES TO SUN CENTER:

   i=complex(0,1)
   vis1=vis0*exp(-i*(alpha)) ; This may be the wrong place to do this.
;                            ; Probably it should be done in calibevents2vis

;  HERMITIANIZE THE VISIBILITIES -- USE ONLY TIMERANGE=[0,2] SECONDS
   vis=vis1
   num_bins=n_elements(vis)
   vis(num_bins/2:num_bins-1)=conj(vis(0:num_bins/2-1))

if keyword_set(plot) then BEGIN
  !p.multi=[0,1,2]
  window,4,xsize=1000,ysize=400
  plot,ce.roll_angle,ce.count,xsty=1,tit='COUNT RATE',xtit='ROLL ANGLE (radians)
  plot,ce.roll_angle,vis0,tit='Visibility (vis0)',xsty=1,xtit='ROLL ANGLE (radians)
  !p.multi=0
  wset,0
endif

; THEN CREATE AN ANNULAR DIRTY MAP:
       mapwidth=256
       dmap_xy=1
       message,'Calling vis2pmap...',/continue
     if keyword_set(plot) then begin
          message,'Calling vis2pmap with /plot set',/cont
          dmap=vis2pmap(vis,ce,det_index,mapwidth ,dmap_xy,/plot)
     endif else begin
          message,'Calling vis2pmap with /plot NOT set',/cont
          dmap=vis2pmap(vis,ce,det_index,mapwidth ,dmap_xy)
     endelse


return,dmap

end
```

vis2pmap.pro

```
function vis2pmap,vis,ce,det_index,map_width,dmap_xy,plot=plot
;+
; NAME: VIS2PMAP
;
; PURPOSE:  Converts visibilities into a polar coordinate map
;
;
;
; METHOD:
;      Compute the convolution of visibility and exp(i*k*r*cos(roll_angle))
;
;
; INPUTS:
;       vis=complex array of N_vis values for phi=0 to 2*!pi
;       Note that the vis must be computed with phase center at the spin axis.
;       ce=calib_eventlist structure
;       det_index=0-27
```

```
;       map_width = scalar width (and height) of map to be made
;
; OUTPUTS:
;       dmap(j_r,n_theta)  = "dirty" map
;       Lots of plots if optional keyword plot is set
;       Optional return of rectangular map if dmap_xy exists
;
; EXAMPLE:
;       N=512
;       roll_angle=findgen(N)*2*!pi/N
;       det_index=4
;       pitch=((hsi_grid_parameters()).pitch)(det_index)
;       i=complex(0,1)
;       vis=100*exp(i*2*!pi*605.*cos(roll_angle)/pitch)
;       count=100+float(vis)
;       phase_map_ctr=2*!pi*600.*cos(roll_angle)/pitch
;       ce={roll_angle:roll_angle,count:count,phase_map_ctr:phase_map_ctr}
;       map_width=200
;       dmap=vis2pmap(vis,ce,det_index,map_width,/plot)
;
;
; FUNCTIONS CALLED:
;      convl.pro
;
; RESTRICTIONS:
;       Makes only annular maps at present
;       The visibilities must have been  phase shifted to the polar center

;
; VERSION HISTORY:
;        VERS. 1 EJS Mar 29, 1999
;        VERS. 2 EJS Sept 28, 1999
;           Doubled width of annulus, increased final map to 360-deg coverage.
;        VERS 3 EJS Oct 15, 1999
;           Replaced spin radius by a call to phase2radius
;           fixed an error in r2
;        VERS 4 EJS Oct 26, 1999
;           Changed Bessel function calls to convl calls.
; NOTES:
;-
; function vis2pmap,vis,ce,det_index,map_width,dmap_xy,plot=plot

message,'Beginning...',/continue

!p.multi=[0,1,2]
; 1. Look at the visibilities and compute the max spatial freq therein.
print,'Look at the visibilities and compute the max spatial freq therein.'
;

roll_angle=ce.roll_angle
phase_map_ctr=ce.phase_map_ctr
sigma=-1.
radius=phase2radius(ce,det_index,sigma)
message,'Deduced spin radius from phase_map_ctr: ='+string(radius),/cont
print,'sigma=',sigma
r0=fix(radius+0.5)          ; radius of the phase center
r1=(r0-map_width)>1 & r2=r1+2*map_width  ; min and max radii to be mapped
if (r1 LE 0) then begin
  message,'r1 LE 0'
endif
```

```
phi1=-!pi & phi2=!pi                    ; min and max azimuths to be mapped
if (keyword_set(plot)) then begin
   plot,float(vis),xsty=1,tit='Real Part of Visibilities'
   plot,imaginary(vis),xsty=1,tit='Imaginary Part of Visibilities'
endif
pitch=((hsi_grid_parameters()).pitch)(det_index)

if (max(abs(vis)) LT .0001) then message,'All visibilities are essentially 0'



Nroll=n_elements(roll_angle)
k=2*!pi/pitch
r=r1+findgen(r2-r1)
kr=k*r
i=complex(0,1)
dmap=fltarr(Nroll,r2-r1)
message,'Doing'+string(r2-r1)+' radii...',/cont

for j=0,r2-r1-1 do begin
  v=exp(-i*kr[j]*cos(roll_angle))
        ; if ((j mod 10) eq 0) then plot,v,tit='r='+string(j+r1),xsty=1
        ; wait,0.1
  d=convl(v,vis)
  dmap[*,r[j]-r1]=d   ; 1st arg is azimuth, 2nd is radius
endfor


if (keyword_set(plot)) then begin
  erase
  ndx=n_elements(dmap(*,0))  & ndy=n_elements(dmap(0,*))
  window,1,xsize=ndx,ysize=ndy,title='Real Part of DMAP'
  print,'Pausing for 1 sec..' & wait,1
  erase

  tvscl,dmap

endif

; Modified Oct 26, 99
  message,'Calling pmap2xy.pro...' ,/continue
;              dmap: 1st index is azimuth (0 to 2*!pi), 2nd is  radius

  dmap_xy=pmap2xy(dmap,r1,r2)
  message,'r1,r0,r2='+string(r1)+string(r0)+string(r2),/continue

if (keyword_set(plot)) then begin
  if (!d.name NE 'PS') then window,0,xsize=960,ysize=960
  tvscl,dmap_xy
endif

message,'Done. Returning',/cont


return,dmap
end
```

pmap2xy.pro

```
function pmap2xy,polmap,r1,r2
; NAME pmap2xy
;
; PURPOSE:
;        Finds the x,y points inside an annulus of radii r1, r2, converts
;        to polar coordinates and uses interpolation to extract values from
;        a polar map polmap of size (
; INPUTS:
;   polmap:
;    1st index is azimuth (0 to 360 degrees)
;    2nd index of polmap is radius (asec)
;   r1,r2 = integer values of inner and outer radii covered by pol
;
; OUTPUT:
;     a square map of size 960 x 960 with 2 asec pixels
;-


x=findgen(960)#replicate(1,960)
y=transpose(x)
r=2*sqrt((x-479.5)^2+(y-479.5)^2)
n_az=n_elements(polmap(*,0))
i_radius=r-r1
j_azimuth=(n_az/2)*(1-atan((y-479.5),(x-479.5))/!pi)
;j_azimuth=(n_az/2)*((2+atan((y-479.5),(x-479.5))/!pi) mod 2)
polmap=float(polmap)

d=interpolate(polmap,j_azimuth,i_radius,missing=0)

return,d


end
```

ls_amp_pha_ctr.pro

```
;+
; NAME: ls_amp_pha_ctr.pro
;
; PURPOSE: Returns the least-square fit of y(angle) to the function
;          A*cos(phase_map_ctr(angle))+B*sin(phase_map_ctr(angle))
;          or equivalently, AMP*cos(phase_map_ctr(angle)-PHA).
;          The result can be used to fit visibilities to HESSI
;          countrates, even in the ranges of slow modulation.
;
; METHOD:
;     Theta=phase_map_ctr
;     Minimize the total squared deviation H of the fit to
;     y[i] = A*cos(Theta[i])+B*sin(Theta[i]))
;
;     H=total{ [y - A*cos(Theta) - B*sin(Theta)]^2 }
;
;    If the angle[i] are chosen to make the cos and sin terms uncorrelated
;    then the regression matrix is diagonal.  If, they are well correlated
;    or nearly anti-correlated, the matrix is singular, and the method fails.
;    The results may be meaningless unless r_p has been
;    chosen such that phi[angle] ranges over a large fraction of [0,2*!pi].
;
```

```
; INPUTS:
;       y=modulation profile (countrates), one for each angle
;       angle=vector of angles (radians), possibly irregular
;       phase_map_ctr=Array of phases of map center relative to line
;          of maximum response, one for each roll angle
;
; OUTPUTS:
;        [A,B] cos,sin amplitudes, OR
;        [amplitude,phase] if optional amp_pha is set,
;        If sigma exists as an argument, the RMS deviation of the fit
;        is returned in sigma.
;        If y has only 2 elements, sigma is necessarily 0 (exact fit).
;        If yfit is a defined argument, fit to y will be returned in it.
;
; EXAMPLE:
;     ;Simple simulation of map-center phases
;     ;Create an array of angles:
;         N=512
;         angle=findgen(N)*!pi/N - !pi/4
;     ;Create some aspect coords:
;         x_asp=1.0*randomn(seed,N)
;         y_asp=1.0*randomn(seed,N)
;     ;Select an inertial map center:
;         X_map=600.
;         Y_map=200.
;     ;Compute the spacecraft coords of map center:
;         x_map_sc = x_asp + X_map*cos(angle) - Y_map*sin(angle)
;         y_map_sc = y_asp + Y_map*cos(angle) + X_map*sin(angle)
;     ;Choose a value for the peak_resp_offset (normally from hessi_grm):
;          peak_resp_offset=!pi/6
;     ;Pick a sub_collimator:
;          ang_pitch=39.45 & det_index=4
;          grid_angle=0.    & h=1
;     ;Now make the array of map-center phases:
;
;     phase_map_ctr=2*!pi*h*(x_map_sc*cos(grid_angle)+y_map_sc*sin(grid_angle)$
;         +peak_resp_offset)/ang_pitch
;     ;Select a range of indices:
;          inx=180+findgen(20)
;           y=cos(phase_map_ctr)+0.25*randomn(seed,N)
;     z=ls_amppha_pmc(angle(inx),y(inx),phase_map_ctr(inx),yfit,sig)
;           plot,angle,y,psym=10,xsty=1 & oplot,angle(inx),yfit,thick=3
;
; RESTRICTIONS:
;       If the data set is enormous, this is slow.
;       For a way to speed it up,
;       see Press and Rybicki, Ap.J., 338, 277, 1989.
;
;         angle must =0 or !pi where the modulation is slowest, and
;         angle must= +!pi/2 or -!pi/2 where modulation is fastest
;
; REVISION HISTORY:
;  Version 1.0, Derived from ls_coscos, EJS, June 15, 1999.
;  Version 2, Added double and bad-fit flag, EJS Sept 30, 1999
;  NOTE: if the sine vector is nearly constant, a bad fit results.
;   This is because the sine is at an extremum and the cosine is
;   very small, so it has to cover all the y variability, and its
;   coefficient will be large. (Similarly if cosine = const.)
;-

function ls_amp_pha_ctr,angl,y,phase_map_ctr,yfit,sigma, $
```

```
                AMP_PHA=amp_pha,verb=verb,double=double,flag=flag
; message,'stopping'
; Compute the sine and cosine arrays

if keyword_set(double) then begin
 if keyword_set(verb) then message,'Going to double precision',/continue
 phase_map_ctr=double(phase_map_ctr)
endif

cosine=cos(phase_map_ctr)
sine=sin(phase_map_ctr)
if (keyword_set(verb)) then message, $
   'Working with '+String(n_elements(phi))+' angles',/cont
            ; The regression matrix
m00=total(cosine^2)  & m01=total(sine*cosine)
m11= total(sine^2)

del=m00*m11-m01^2
if abs(del) lt 1.e-2*(abs(m00)+abs(m11)) then begin
  message,'Ill-conditioned LS fit.',/continue
  ;message,string(del)+string(m00)+string(m01)+string(m11),/continue
  flag=1
endif
c0=total(y*cosine)
c1=total(y*sine)

A=(c0*m11-c1*m01)/del   ; Solution to M # [A,B] = [c0,c1]
B=(c1*m00-c0*m01)/del

AB=[A,B]

yfit=A*cosine+B*sine   ; = amp*cos(phase_map_ctr-pha)
; compute the goodness of fit
if (n_elements(sigma) GT 0) then sigma=sqrt(variance(y-yfit))


if keyword_set(amp_pha) then AB=[sqrt(a^2+b^2),atan(B,A)]

return,AB

end
```

phase2radius.pro

```
function phase2radius,ce,det_index,rmsdev
;+
; NAME: phase2radius
;
; PURPOSE:
;
; Finds the average spin radius from the pitch and the
; calib_eventlist tags .phase_map_center and .roll_angle
; for detector det_index
;
; INPUTS:
;         ce=calib_eventlist structure
;         det_index= scalar (0-27)
;
; OUTPUTS:
;         radius = scalar FLOAT
```

```
;
; VERSION HISTORY:
;   Upgraded inputs to structure ce
;                       EJS Oct. 26, 1999
;-

 pitch=((hsi_grid_parameters()).pitch)(det_index mod 9)
 if n_elements(rmsdev) NE 0 then rmsdev=-1.0

 yfit=1.
 z=ls_sin_cos(ce.roll_angle,ce.phase_map_ctr,1,yfit,rmsdev)
 radius=sqrt(z(0)^2+z(1)^2)*pitch/(2*!pi)

return,radius

end
```

convl.pro

```
FUNCTION CONVL,A,P
;+
;  PURPOSE:
;   IDL routine to convolve array A with a point spread function P
;
;  CALLING SEQUENCE:   C=CONVL(A,P)
;
;  INPUTS:
;    A = map vector or array (REAL OR COMPLEX)
;    P = PSF vector or array, same size as A
;
;  OUTPUT:
;    Convolved matrix or vector, shifted to make the origin the same
;    as A.  If P is 1 at array center, C=A
;
;  RESTRICTIONS:
;    A and P must be vectors or matrices of the same size, dimension LE 2
;
;  NOTES:
;    The astron program "convolve" does not allow input arrays of
;    equal size, so this program is complementary to it.
;    With this function,
;        total(C)=total(A)*total(P)
;        convl(A,P)=convl(P,A)
;        convl(A,P1+P2)=convl(A,P1)+convl(A,P2)
;
;  VERSION HISTORY:
;    VERS. 2.0 OCTOBER 25, 1999, EJS
;
;-

sz_a=size(a) & sz_p=size(p)
if (sz_a(0) NE sz_p(0)) then message,"Input arrays don't have same dimensions"
if (sz_a(0) eq 1) then $
   if (sz_a(1) NE sz_p(1)) then message,'Input vectors have different sizes'
if (sz_a(0) eq 2) then begin
   if ( (sz_a(1) NE sz_p(1)) OR (sz_a(2) NE sz_p(2)) ) then $
                message,'Input arrays have different sizes'
   n1=sz_a(1) & n2=sz_a(2)
endif
if (sz_a(0) gt 2) then message,'Arrays of dimension 3 or higher not supported'
```

```
c=float(fft(fft(a,1)*fft(p,1),-1) )

if sz_a(0) eq 1 then c=shift(c,sz_a(1)/2) $
else            c=shift(c,sz_a(1)/2,sz_a(2)/2)


return,c

end
```

Edward J. Schmahl
Last modified: Nov 8 09:55 EDT 1999

SECTION III
HESSI Imaging in the Sparse-Photon Limit

HESSI IMAGING IN THE SPARSE PHOTON LIMIT

Aschwanden and Schmahl, 2000 (Partial preprint)

INTRODUCTION

The HESSI telescope is capable of recording solar hard X-ray bursts on a photon-by-photon basis by determining the time of photon arrival to less than 0.5 millisecond. The HESSI imaging team has made simulations that indicate that maps can be constructed from as few as ~ 100 photons, provided that the background flux is small. l is important to make estimates of goodness of fit to models based on such sparse data, and to assess the validity of such models in a mathematically rigorous way. Clearly, histograms of mere "handfuls" of photons do not follow the usual Gaussian distributions, and one must go to more basic distributions. The following is a sketch of the statistical methods we plan to employ for HESSI sparse-photon imaging.

χ^2 and C-statistic for Sparse Sampling

An observed time series of photon counts, $f^{obs}(t)$, which is binned into time intervals of length Δt, so that the number $n_i = f^{obs}[t_i < t < t_i + \Delta t]$ represents the photon counts in bin $[t_i, t_i + \Delta t]$, is modeled by an analytical function, $f^{model}(t)$, which predicts an expectation value $e_i = f^{model}[t_i < t < t_i + \Delta t]$ of photon counts in time bin i. In the limit of large-number counts, $n_i \gg 0$, Poisson statistics can be approximated by a Gaussian distribution, which predicts an r.m.s. noise of $\sigma_i = \sqrt{e_i} \approx \sqrt{n_i}$. The goodness of fit of a model can then be evaluated by the well-known χ^2-statistic,

$$\chi^2 = \frac{1}{N}\sum_{i=1}^{N}\frac{(n_i - e_i)^2}{\sigma_i^2} = \frac{1}{N}\sum_{i=1}^{N}\frac{(n_i - e_i)^2}{e_i}, \qquad (1)$$

yielding a normalized value of $\chi^2 \approx 1$ in the case when the model is consistent with the data, within the uncertainties of the expected noise.

For sparse sampling, in particular when the observed time series n_i contains zeroes or few counts, say $n_i \lesssim 10$, e.g. as it is the case for the modulation profiles of the finest HESSI collimators, where the total photon counts are binned into some 2^{12}=4096 time bins to resolve the rotational modulation, the Gaussian χ^2-statistic is not appropriate anymore. Photon counting statistics for sparse sampling can be derived from the general probability function P as outlined by Cash(1979), which is

$$P = \prod_{i=1}^{N} \frac{e_i^{n_i} e^{-e_i}}{n_i!} \tag{2}$$

for a particular result n_i given the correct set of e_i. The likelihood radio, called *C-statistic* by Cash (1979), can be expressed in logarithmic form from Eq.A2,

$$C_{Cash} = -2 \ln P = -2 \sum_{i=1}^{N} (n_i \ln e_i - e_i - \ln n_i!) \,. \tag{3}$$

Let the theoretical model $e_i(\Theta_1, ..., \Theta_p)$ be defined by p free parameters $\Theta_i, i = 1, ..., p$. The best-fitting model is found by varying all p parameters $\Theta_1, ..., \Theta_p$ until the C-statistic reaches a minimum, which we denote by $(C_{min})_p$. Now, during the iterative fitting procedure, only a partial subset of model parameters, say q parameters (with $q < p$), may have already converged to the true solution, $\Theta_1^T, ..., \Theta_p^T$, so that $\Theta_1, ..., \Theta_q$ are set to $\Theta_1^T, ..., \Theta_q^T$, while the remaining p-$q$parameters, $\Theta_{q+1}, ..., \Theta_p$, still need to be varied until a global minimum of C is reached. We denote this partial solution, where p-q parameters have already converged to the true solution Θ_i^T, with the value $(C_{min})^T{}_{p-q}$. According to the theorem of Wilks (1938; 1963), the difference,

$$\Delta C = (C_{min})_{p-q}^T - (C_{min})_p \tag{4}$$

will be distributed as χ^2 with q degrees of freedom. Therefore, the quantity ΔC can be

used to establish a confidence criterion of the model e_i to the data n_i. Following Cash (1979), the term $\ln(n_i!)$ of Eq.A3 drops out in the difference ΔC, because only the parameters $e_i(\Theta_1, ..., \Theta_p)$ are varied during the fitting procedure. Thus, it is more convenient to use the simplified statistic

$$C_{simplified} = -2\ln P = -2\sum_{i=1}^{N}(n_i \ln e_i - e_i) \ . \tag{5}$$

For the evaluation of the difference ΔC in Eq.A4 we have the partially optimized term $(C_{min})^T{}_{p\text{-}q}$,

$$(C_{min})^T_{p-q} = -2\sum_{i=1}^{N}(n_i \ln (e_i)^T_{p-q} - (e_i)^T_{p-q}) \ , \tag{6}$$

and the absolute minimum $(C_{min})_p$, which represents the asymptotic limit when the data n_i perfectly match the model, i.e. e_i=n_i,

$$(C_{min})_p = -2\sum_{i=1}^{N}[n_i \ln(n_i) - n_i] \ . \tag{7}$$

The combination of Eq.A4-A6 yields then

$$\Delta C = -2\sum_{i=1}^{N}[n_i \ln(e_i) - e_i - n_i \ln(n_i) + n_i] \ . \tag{8}$$

Instead of the reduced χ^2-statistic (A1), we can therefore use the equally-simple

C-statistic (where we drop the symbol Δ for brevity),

$$C := \frac{1}{N}\Delta C = \frac{2}{N}\sum_{i=1}^{N}[n_i \ln(\frac{n_i}{e_i}) - (n_i - e_i)] \; . \qquad (9)$$

This form has the advantage that, in addition to being asymptotic to χ^2, C vanishes identically when the model fits the data exactly. Numerical care has to be taken for the time bins that contain zeroes in the data, n_i=0, in which case the mathematical relation $n_i \ln(n_i) = 0$ has to be used to avoid the singularity $\ln(n_i) \mapsto \infty$. On the other side, a singularity could arise when the model predicts zero counts, e_i=0, but the observed counts are not zero, $n_i \neq 0$, because the term $-n_i \ln(e_i)$ yields than infinity. It seems therefore to be recommendable to restrict the model fit to time intervals with a finite probability for photon counts, i.e. e_i>0.

Ed Schmahl
1999-12-17